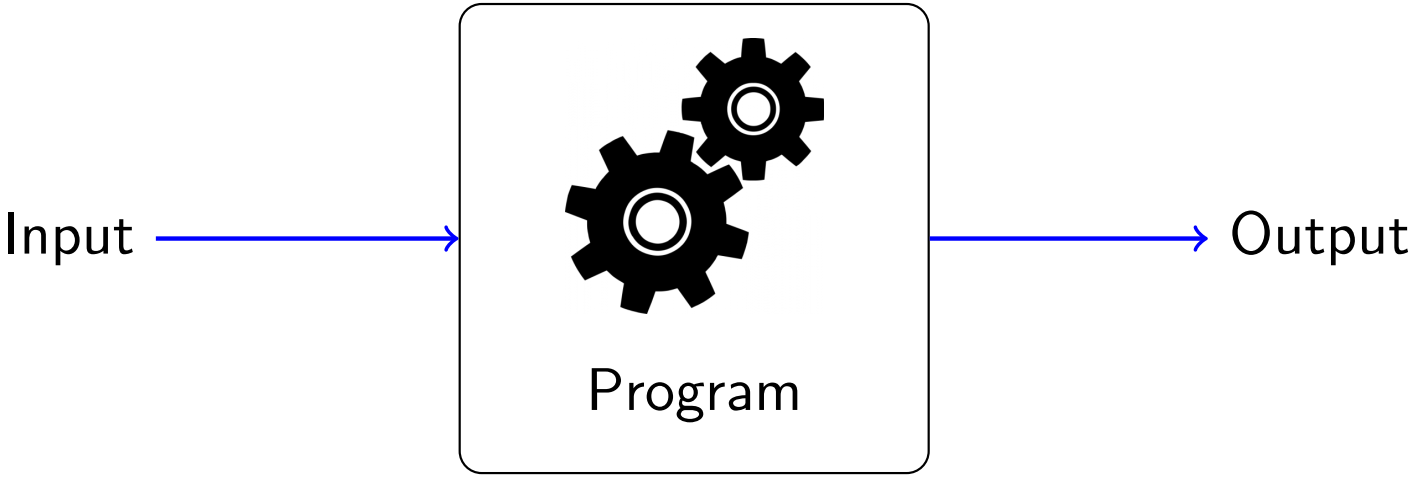
A Unified Image Processing Framework for Computer Vision and Remote Sensing

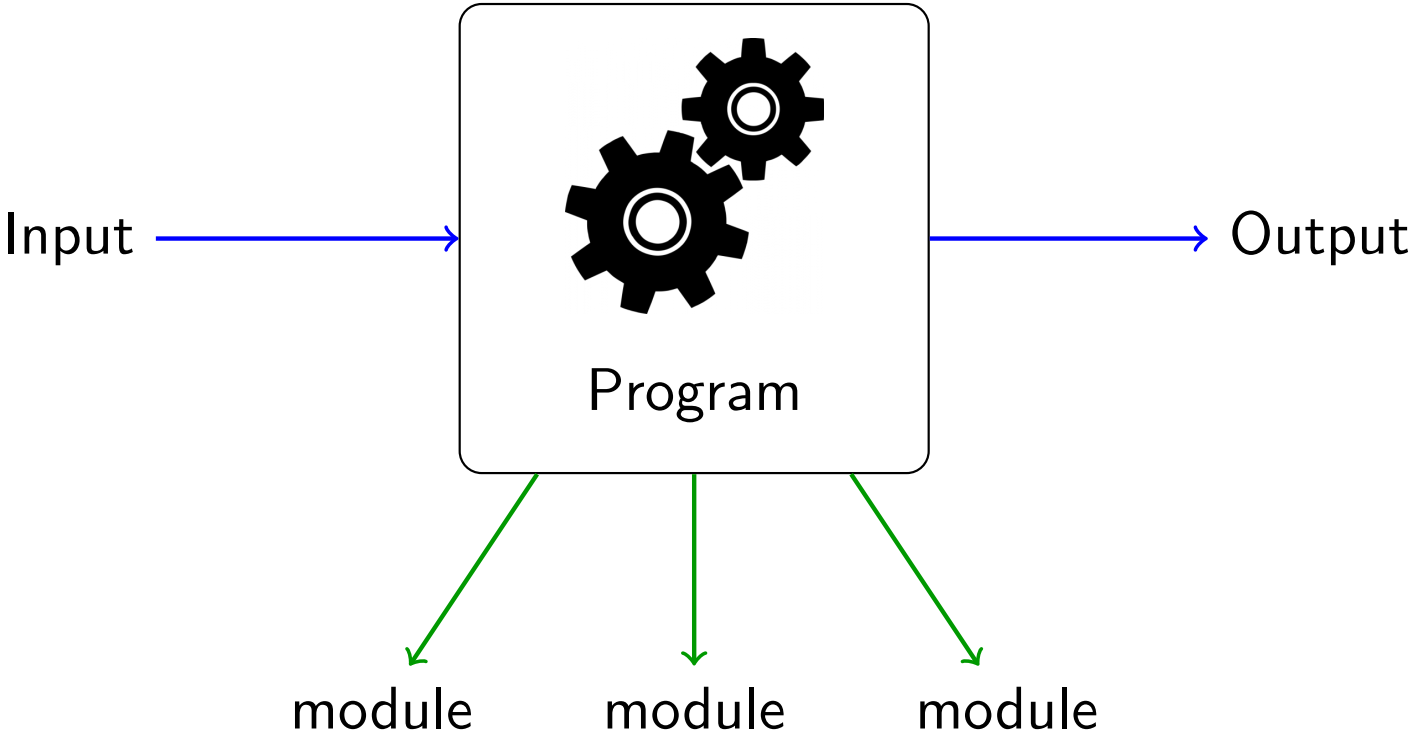
Carsten Brandt, Ludmilla Brandt, Marcus Zepp, Akarsh Seggemu

July 15, 2015

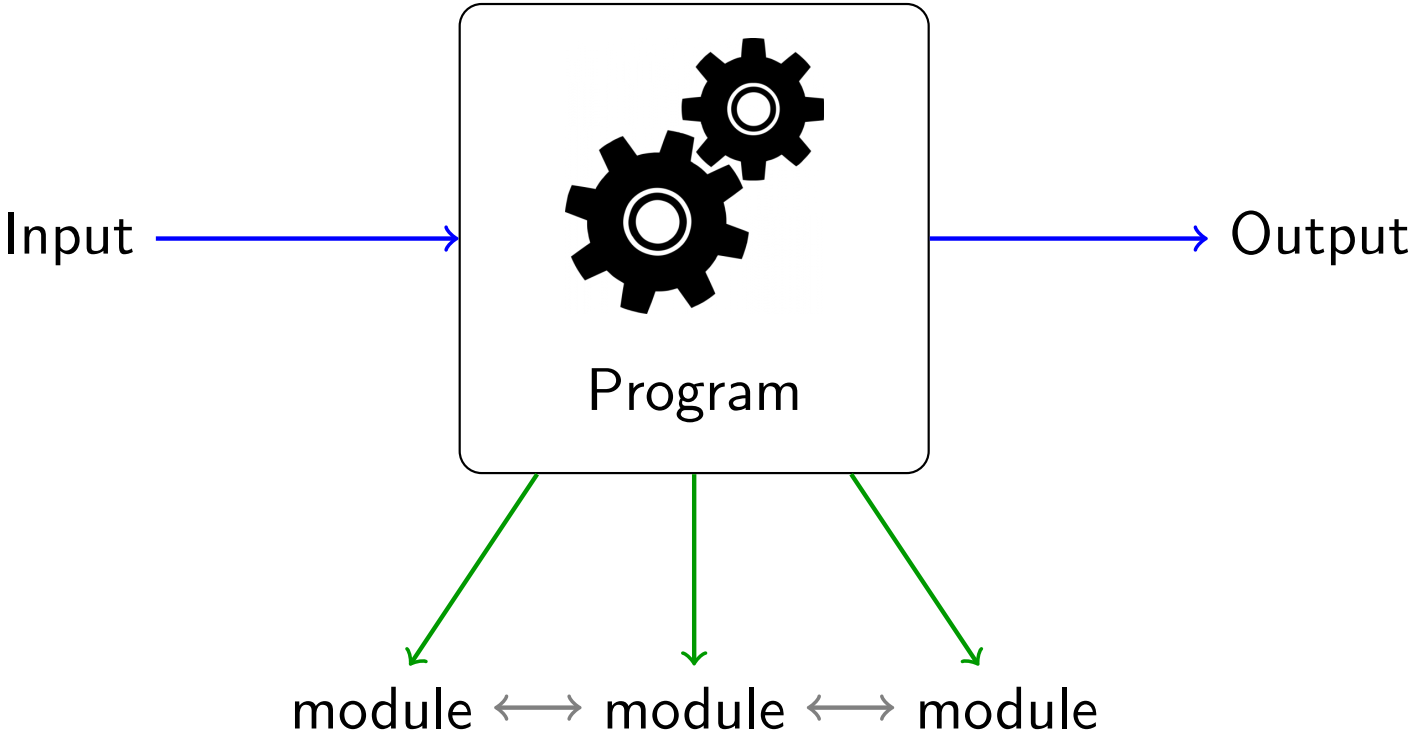
Unified Image Processing Framework



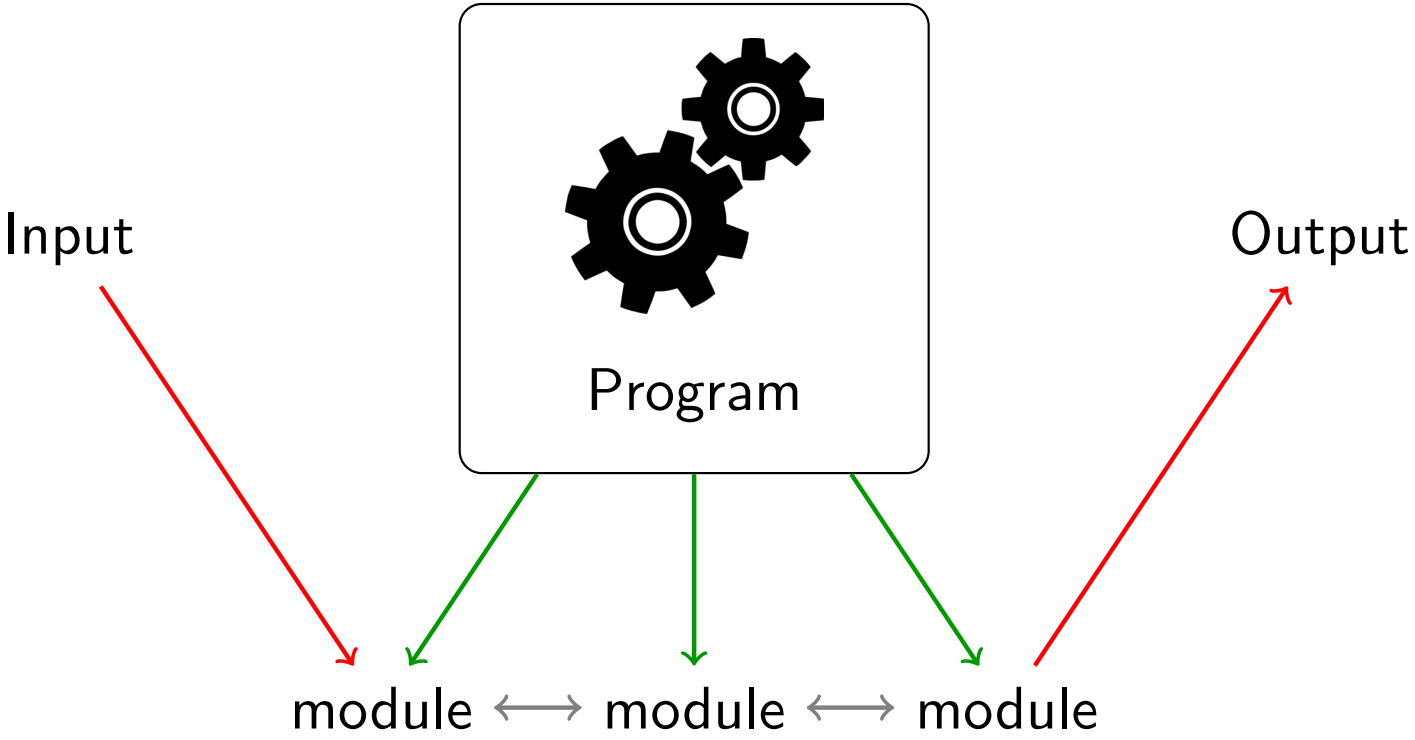
Unified Image Processing Framework



Unified Image Processing Framework



Unified Image Processing Framework



Use given modules

Framework

- module 1
- module 2

⋮

- module n

Specify execution order

Chain

- step 1: module 1
- step 2: module 2
- step 3: module 7
- step 4: module 13

dependencies

Modify the chain

Chain 1

- step 1: module 1
- step 2: module 2
- step 3: module 7
- step 4: module 13

dependencies

Modify the chain

Chain 2

step 1: module 1

step 2: module 2

step 3: module 24

step 4: module 13

dependencies

Modify the chain

Chain 3

step 1: module 1

step 2: module 2

step 3: module 1

step 4: module 13

dependencies

Modify the chain

Chain 4

step 1: module 1

step 2: module 2

step 3: module 1

step 4: module 13

dependencies

Modify the chain

Chain 5

step 1: module 1

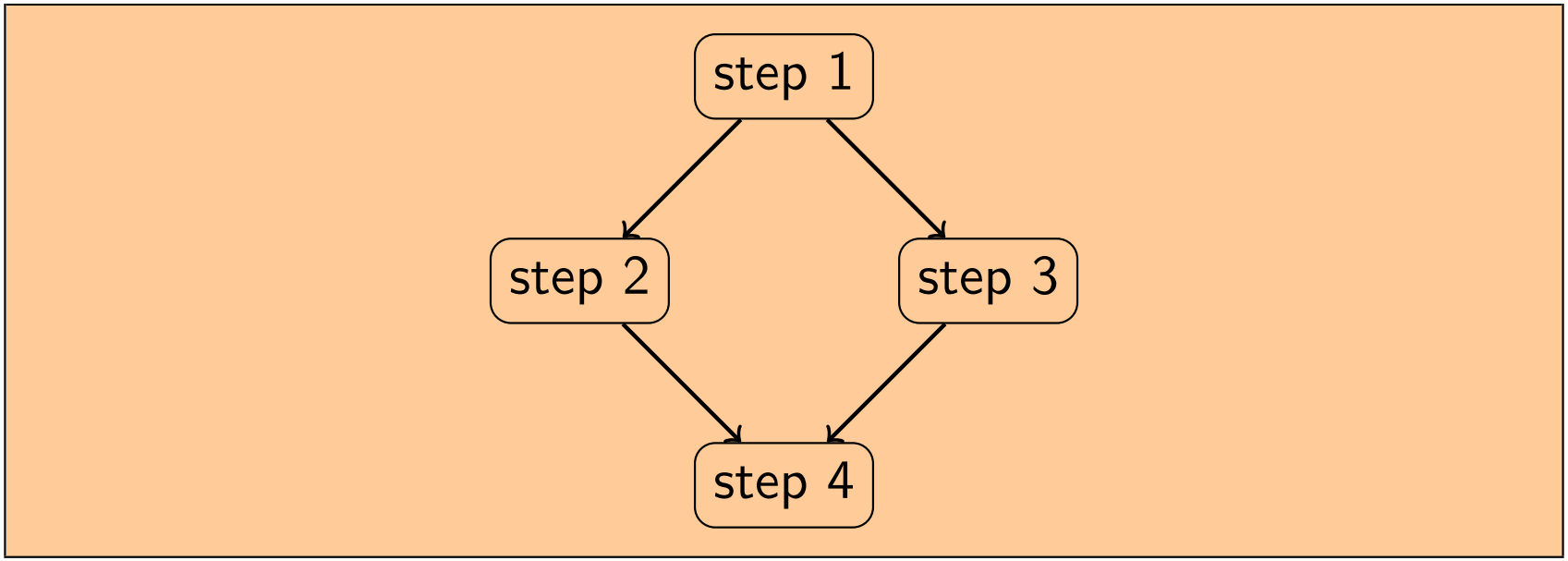
step 2: module 2

step 3: module 1

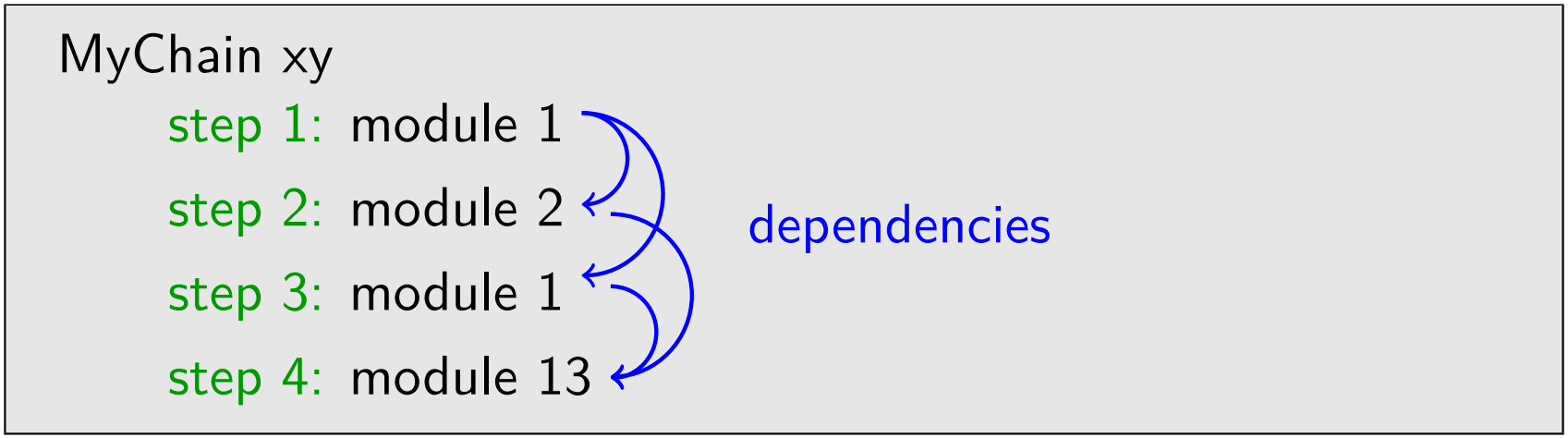
step 4: module 13

dependencies

Visualization



Storage


MyChain xy
step 1: module 1
step 2: module 2
step 3: module 1
step 4: module 13
dependencies

Extend the modules

Framework

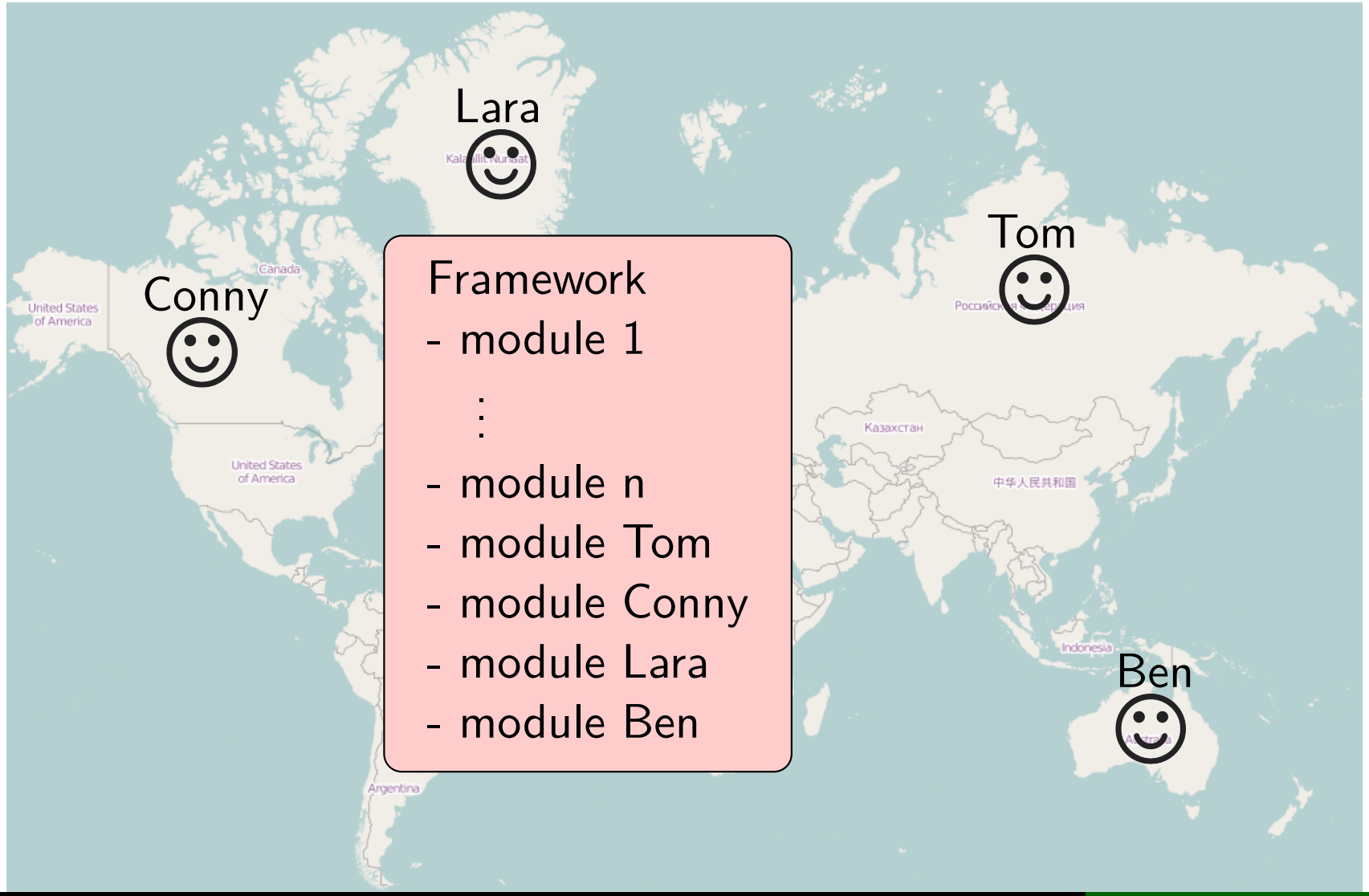
- module 1
- module 2

  ⋮

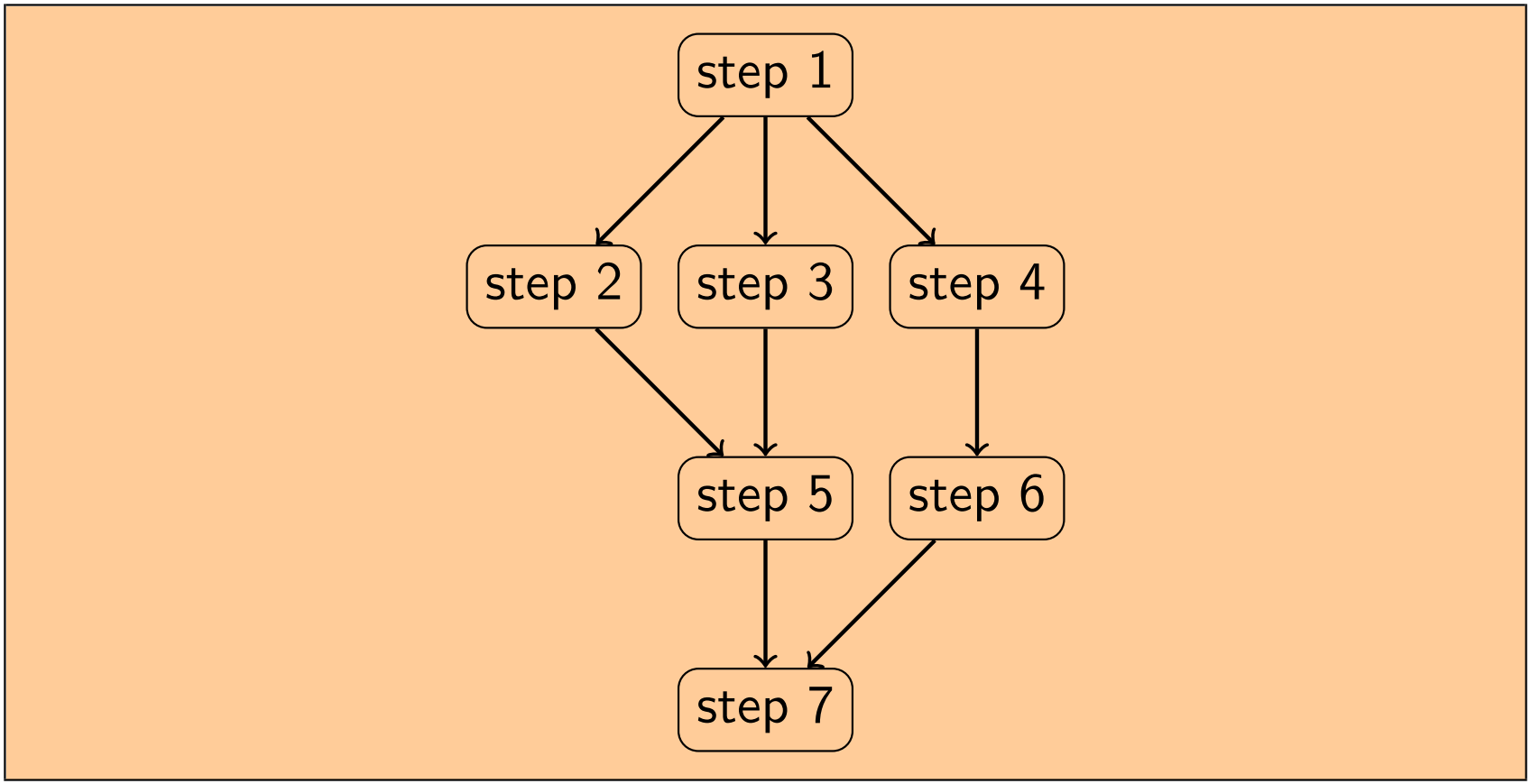
- module n
- module n+1
- module n+2

  ⋮

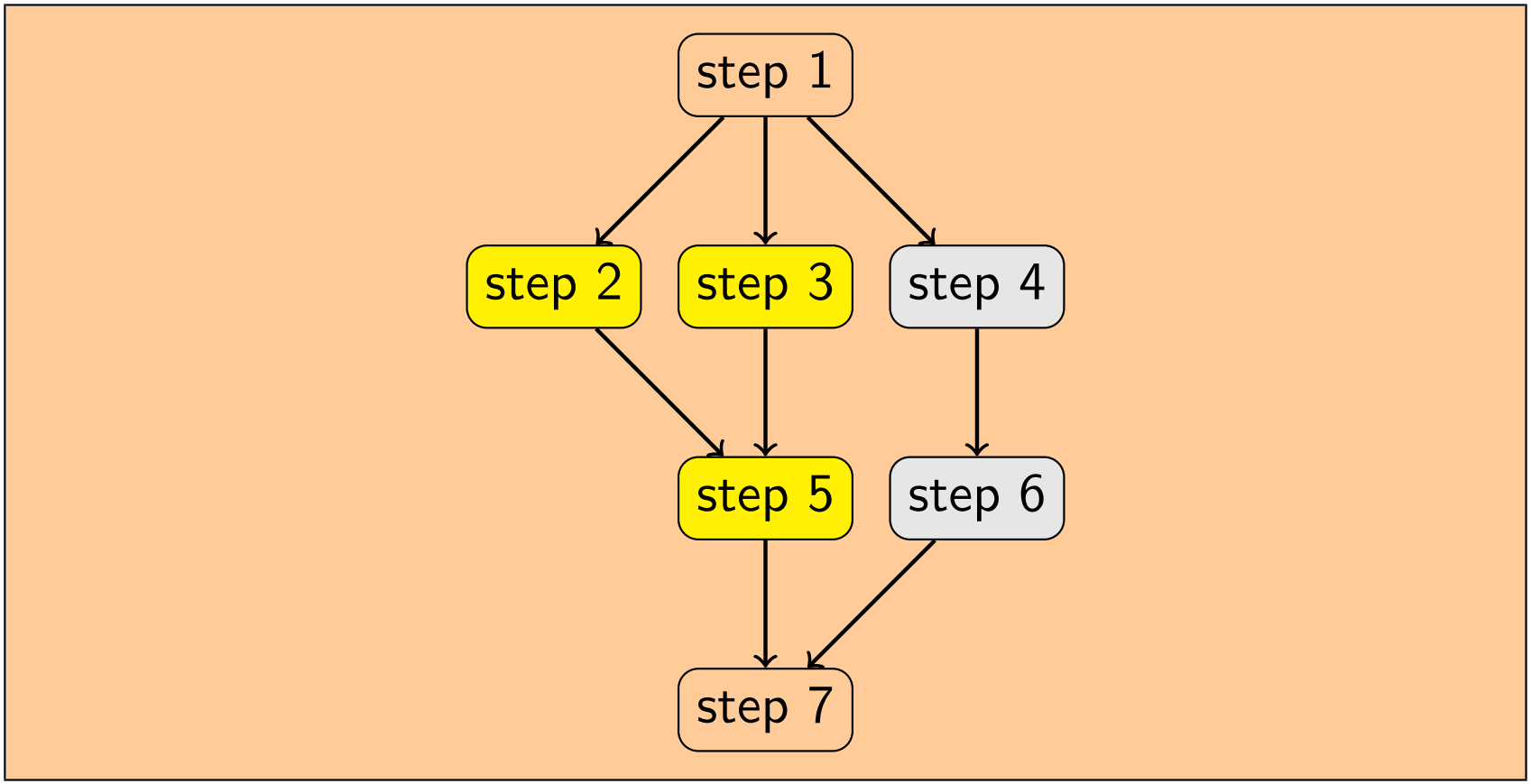
Share the modules



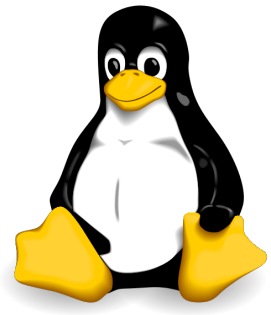
Parallel execution



Parallel execution



Platform independent

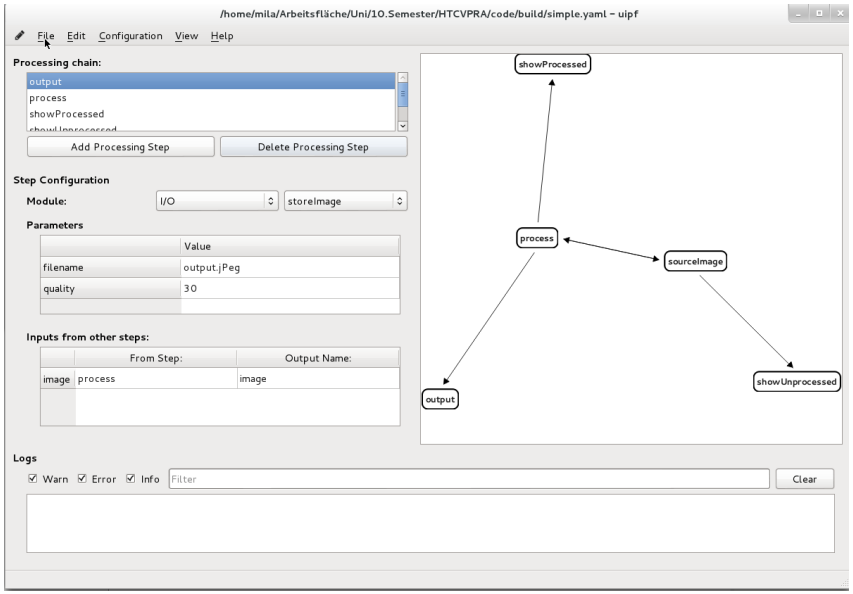


Framework

Microsoft Windows

User Interfaces

GUI



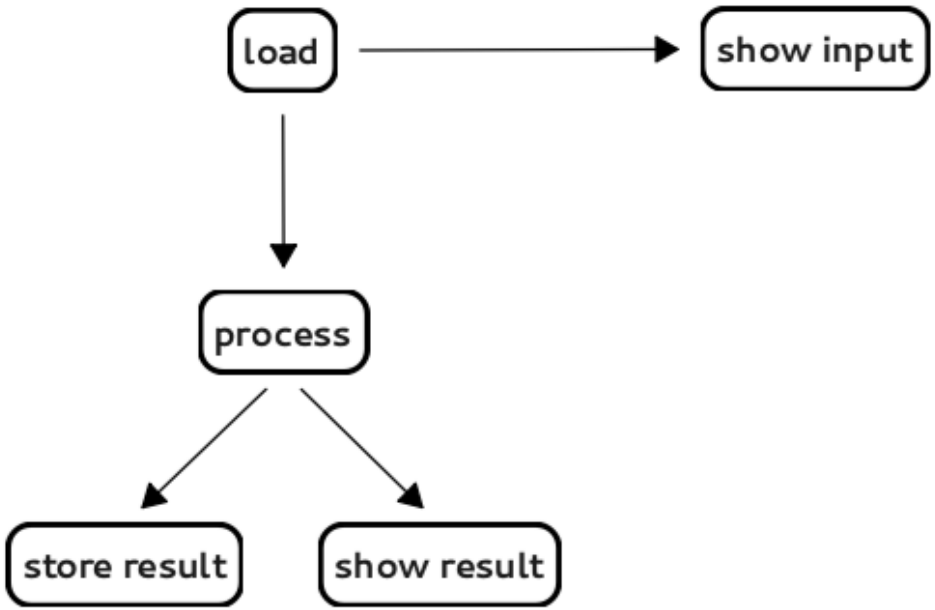
Console

```
Usage:
./uipf -c <path to configuration file>                          run a processing chain
from a config file.
./uipf <moduleName> -i <input> [-p <params> -o <output>]        run a single module.

Allowed options:

Generic options:
  -v [ --version ]      print version string
  -h [ --help ]         produce help message

Program options:
  -c [ --configuration ] arg defines the path to an already created and stored
                             chain configuration, written in yaml
  -i [ --input ] arg         defines an input, can be used multiple times
                             format: inputName:fileName inputName is optional
                             if there is only one input
  -o [ --output ] arg        defines an output, can be used multiple times,
                             format: outputName:fileName. outputName is
                             optional, if there is only one input. Output is
                             optional, if there is only one input and one
                             output, the output filename will be chosen from
                             the input name in this case.
  -p [ --param ] arg         defines a parameter, format:  name:value
```

General Concepts

Processing Chain

→ Processing Step

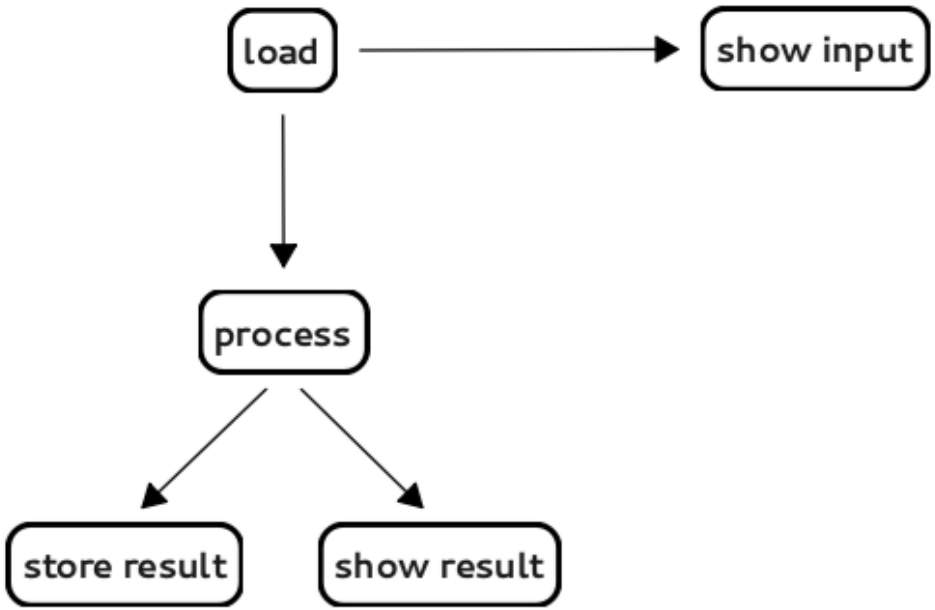


General Concepts

Processing Chain

→ Processing Step

→ Module



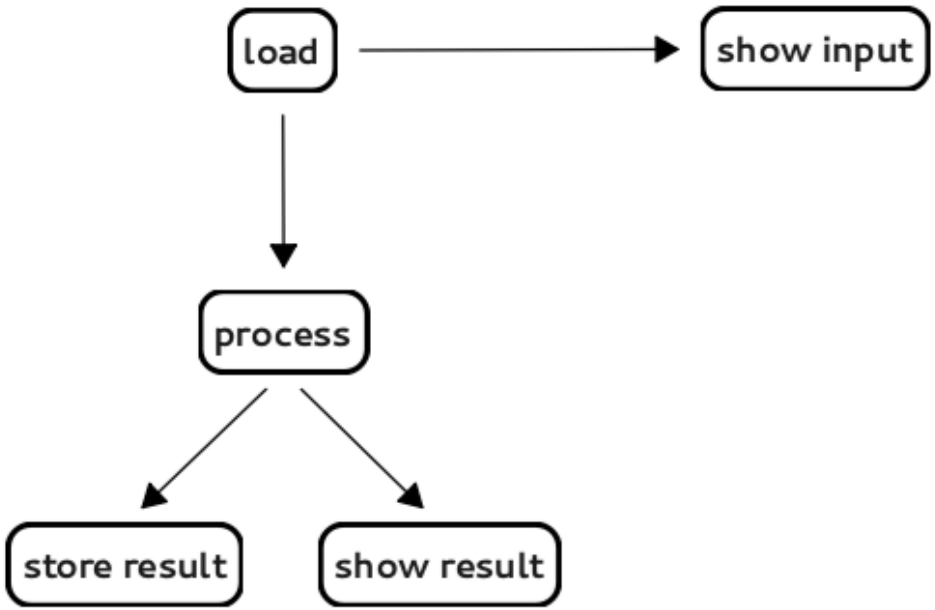
General Concepts

Processing Chain

→ Processing Step

→ Module

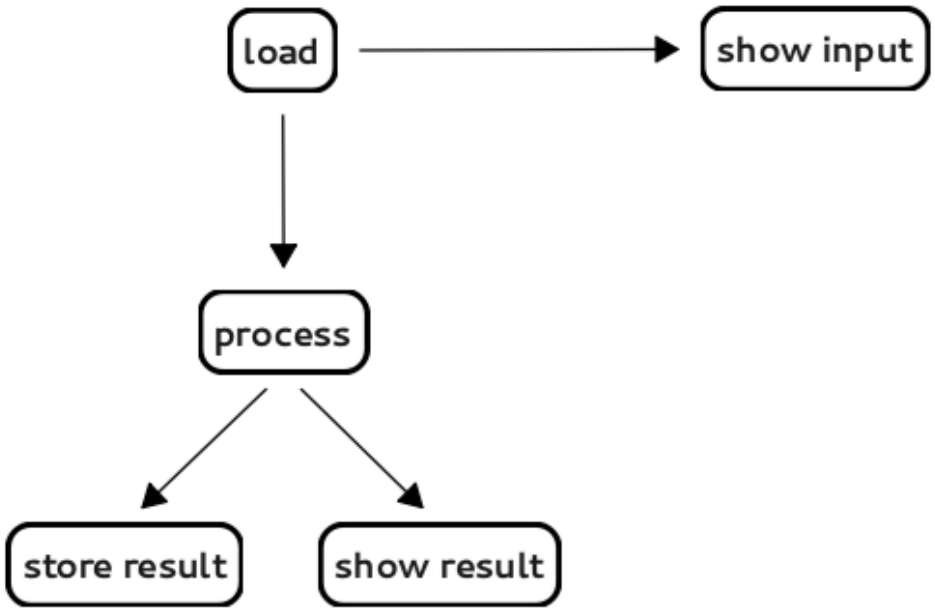
→ Parameters



General Concepts

Processing Chain

- → Processing Step
  - → Module
  - → Parameters
  - → Dependencies

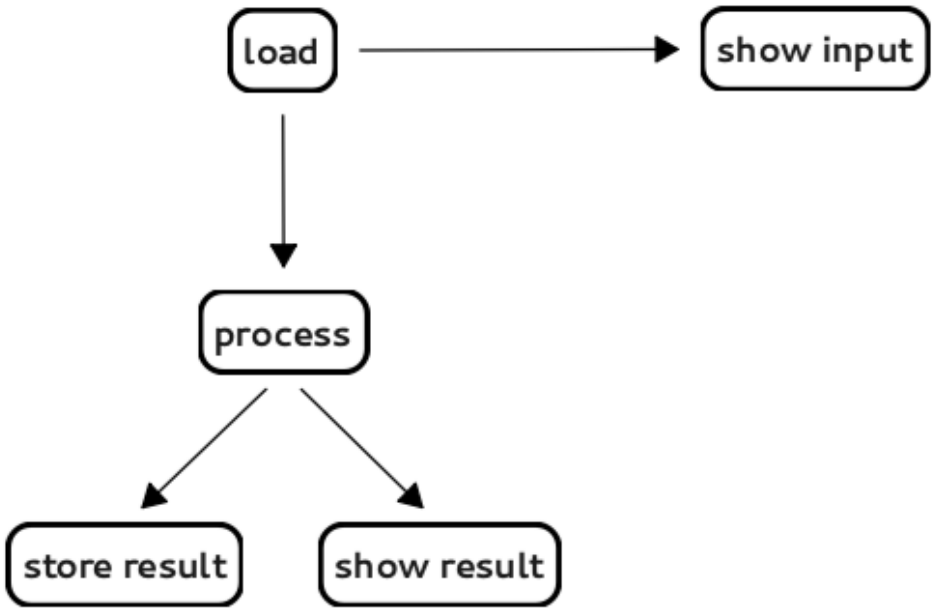


General Concepts

Processing Chain

→ Processing Step

- → Module
- → Parameters
- → Dependencies



Storage format: YAML

```yaml
input:
  module: loadImage
  filename: input.png

gauss:
  module: gaussian
  input:
    image: input.image
  sigmaX: 10
  sigmaY: 15

show:
  module: showImage
  input:
    image: gauss.image
```

Storage format: YAML

```yaml
input:
  module: loadImage
  filename: input.png

gauss:
  module: gaussian
  input:
    image: input.image
  sigmaX: 10
  sigmaY: 15

show:
  module: showImage
  input:
    image: gauss.image
```

Storage format: YAML

```yaml
input:
  module: loadImage
  filename: input.png

gauss:
  module: gaussian
  input:
    image: input.image
  sigmaX: 10
  sigmaY: 15

show:
  module: showImage
  input:
    image: gauss.image
```

Storage format: YAML

```yaml
input:
  module: loadImage
  filename: input.png

gauss:
  module: gaussian
  input:
    image: input.image
  sigmaX: 10
  sigmaY: 15

show:
  module: showImage
  input:
    image: gauss.image
```

Storage format: YAML

```yaml
input:
  module: loadImage
  filename: input.png

gauss:
  module: gaussian
  input:
    image: input.image
  sigmaX: 10
  sigmaY: 15

show:
  module: showImage
  input:
    image: gauss.image
```

Storage format: YAML

```yaml
input:
  module: loadImage
  filename: input.png

gauss:
  module: gaussian
  input:
    image: input.image
  sigmaX: 10
  sigmaY: 15

show:
  module: showImage
  input:
    image: gauss.image
```

The Graphical User Interface

/data/cebe/Dokumente/Uni/htcv-pj-a/examples/convolution.yaml – uipf

File Edit Configuration View Help

Processing chain:

kernel
result
show image
show kernel

Add Processing Step | Delete Processing Step

Step Configuration

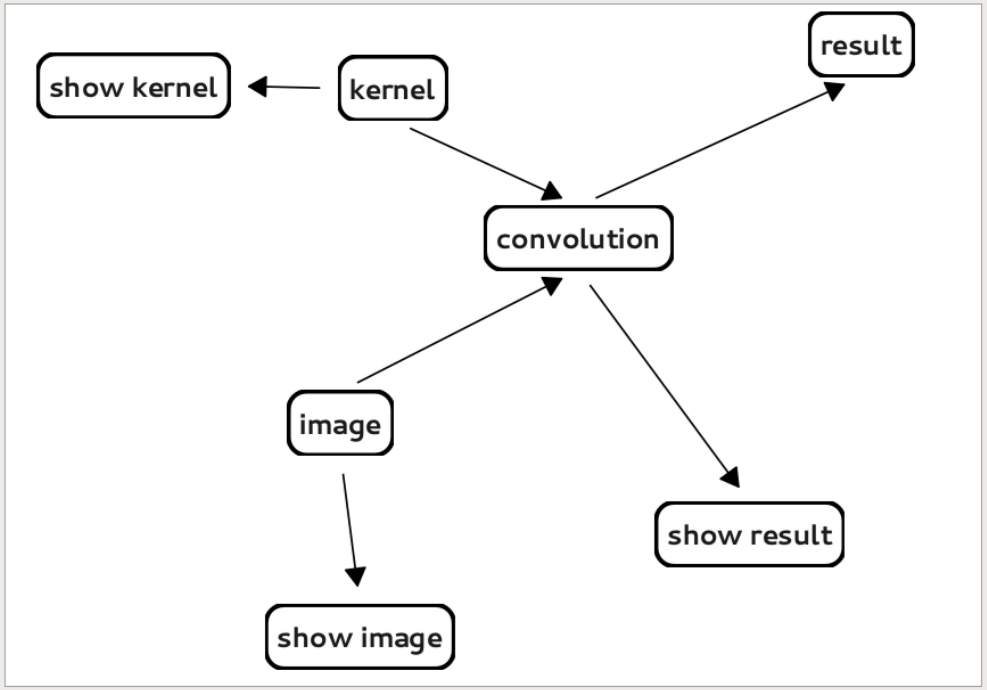
Module: I/O | storeImage

Parameters

	Value
filename	ouput.png
quality	

Inputs from other steps:

	From Step:	Output Name:
image	convolution	image



Logs

☑ Warn ☑ Error ☑ Info Filter Clear

The Graphical User Interface

Processing chain:

- kernel
- result
- show image
- show kernel

Add Processing Step | Delete Processing Step

Step Configuration

Module: I/O | storeImage

Parameters

	Value
filename	ouput.png
quality	

Inputs from other steps:

	From Step:	Output Name:
image	convolution	image

The Graphical User Interface

/data/cebe/Dokumente/Uni/htcv-pj-a/examples/convolution.yaml – uipf

File Edit Configuration View Help

Processing chain:

kernel
result
show image
show kernel

Add Processing Step | Delete Processing Step

Step Configuration

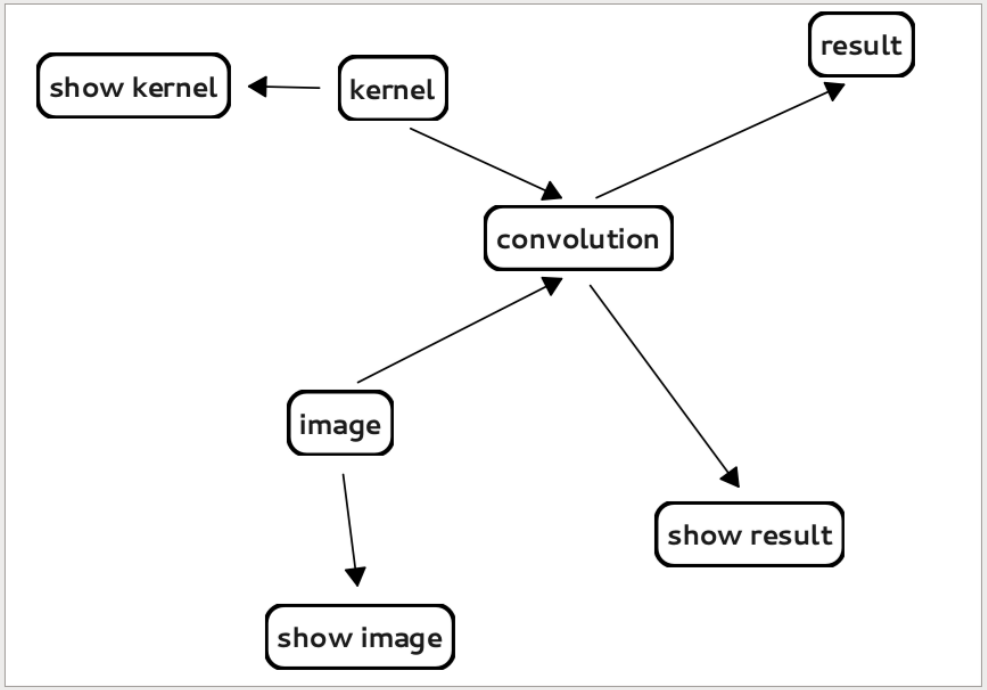
Module: I/O | storeImage

Parameters

	Value
filename	ouput.png
quality	

Inputs from other steps:

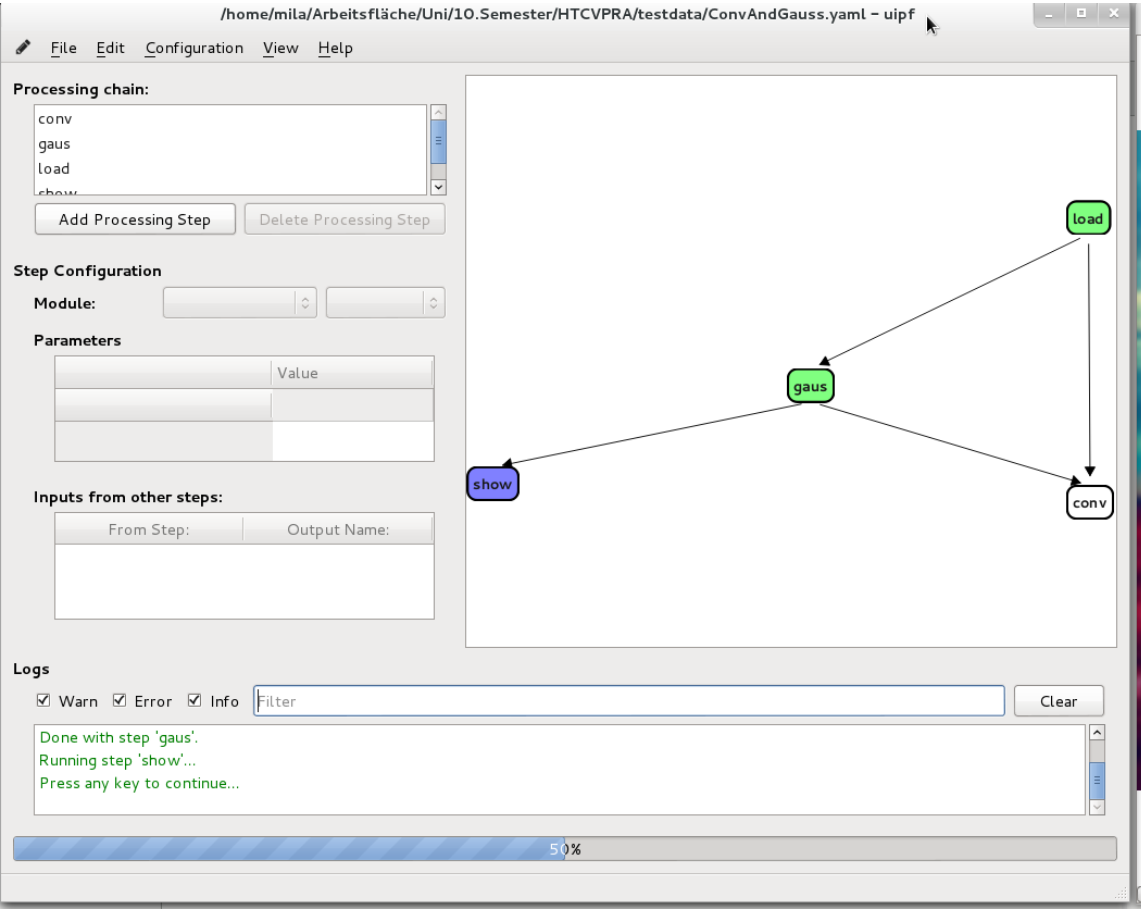
	From Step:	Output Name:
image	convolution	image



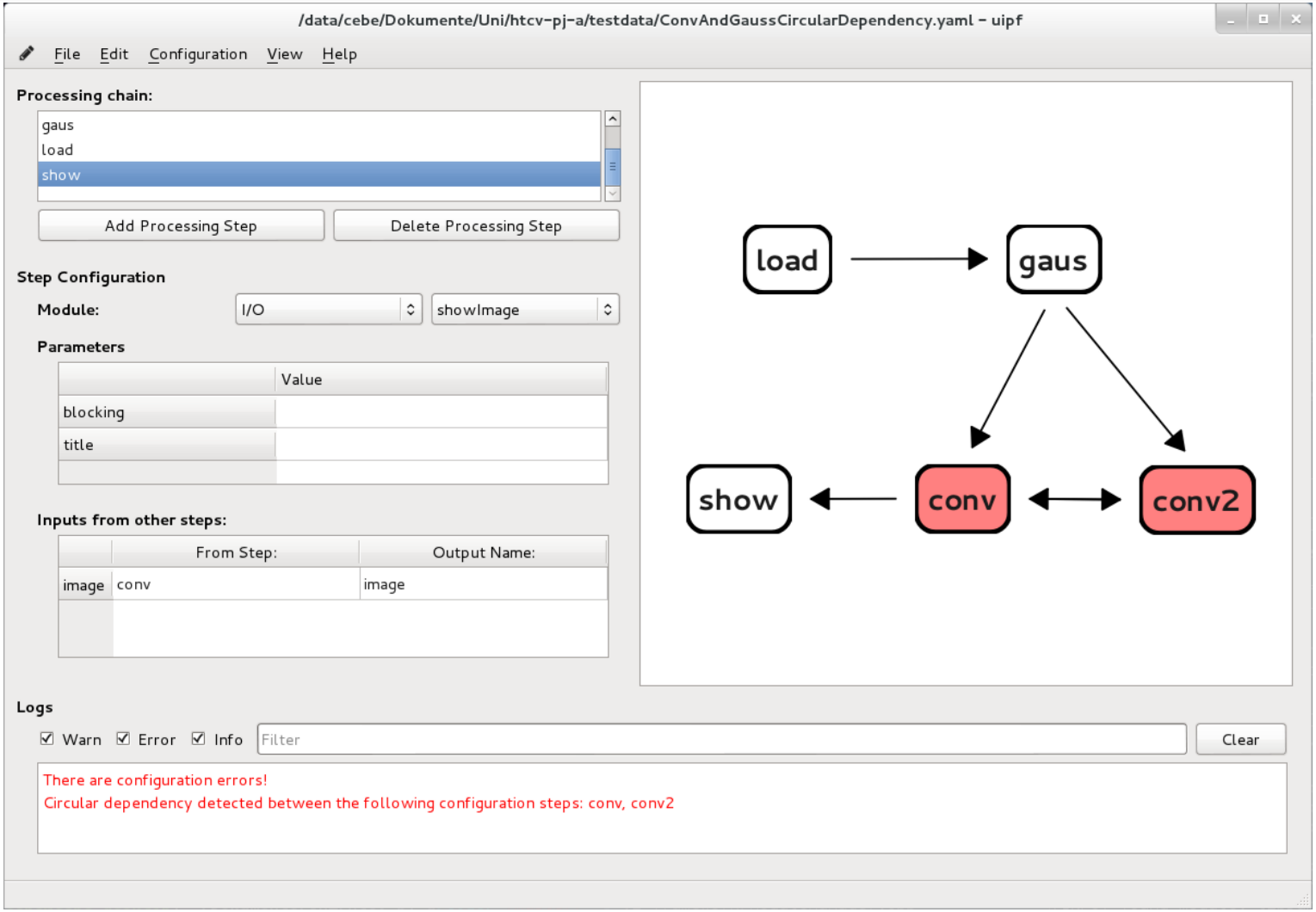
Logs

☑ Warn ☑ Error ☑ Info Filter Clear

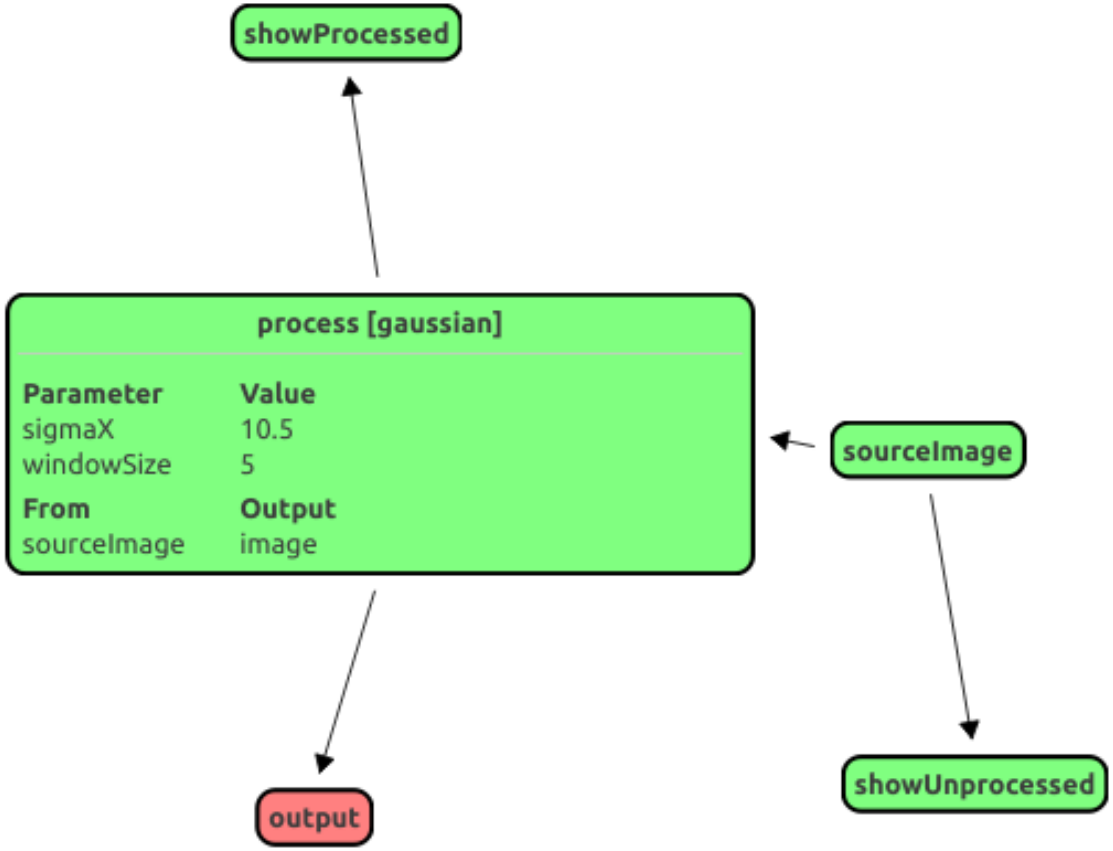
The Graphical User Interface



The Graphical User Interface



Interface features: Graph



The Console Interface

```
./uipf -c myconfig.yaml
```

```
./uipf <moduleName> -i <input>
                    -p <params>...
                    -o <output>
```

The Console Interface

```
./uipf -c myconfig.yaml
```

```
./uipf <moduleName> -i <input>
                    -p <params>...
                    -o <output>
```

The Console Interface

```
./uipf gaussian -i input.jpg -p sigmaX:5
```

```
./uipf gaussian -i input.jpg -p sigmaX:5
                -o notdefault.png
```

The Console Interface

```
./uipf gaussian -i input.jpg -p sigmaX:5
```

```
./uipf gaussian -i input.jpg -p sigmaX:5
                -o notdefault.png
```

```
./uipf convolution -i image:input.jpg
                   -i kernel:kernel.png
                   -p sigmaX:5 -o out.png
```

The Console Interface

```
./uipf gaussian -i input.jpg -p sigmaX:5
```

```
./uipf gaussian -i input.jpg -p sigmaX:5
                -o notdefault.png
```

```
./uipf convolution -i image:input.jpg
                   -i kernel:kernel.png
                   -p sigmaX:5 -o out.png
```

Basics

Modules are precompiled extensions that:

- encapsulate functionality which can be used in processingsteps
- are binary files, shareable without sourcecode (QTPlugin)
- implement a simple interface
- can include own libs as they need them
- have Metadata displayed in the GUI

Interface

Basic interface:

```
string name();
void run(DataManager& data);
MetaData getMetaData();
```

MetaData:

```
string, // general verbal description of the module
string, // category
DataDescriptionMap, // input
DataDescriptionMap, // output
ParamDescriptionMap // params
```

Interface

Basic interface:

```
string name();
void run(DataManager& data);
MetaData getMetaData();
```

MetaData:

```
string, // general verbal description of the module
string, // category
DataDescriptionMap, // input
DataDescriptionMap, // output
ParamDescriptionMap // params
```

run() method

```
1 void LoadImageModule::run( DataManager& data) const
2 {
3 // (1) get inputs and params:
4 //      - data.getInputData(inputName)
5 //      - data.getParam (paramName, dafaultValue)
6 // (2) work with them
7 // (3) create output:
8 //      - data.setOutputData(outputName,
      outputContent);
9 }
```

getMetaData() method

```
MetaData ResizeModule::getMetaData() const
{
    DataDescriptionMap input = {{"image", DataDescription(MATRIX, "the image to resize.") }};

    DataDescriptionMap output = {{"image", DataDescription(MATRIX, "the result image.") }};

    ParamDescriptionMap params = {
        {"width", ParamDescription("new width") },
        {"height", ParamDescription("new height") }
    };

    return MetaData("Resizes an image using openCV.","Image Processing",input,output,params );
}
```

CMake

Modules need to be registered in CMakeList.txt:

```
...
#Image Processing
#ResizeModule
add_library(ResizeModule SHARED modules/improc/ResizeModule.cpp)
qt5_use_modules(ResizeModule Core ) #QtCore is needed for <QPlugin>
target_link_libraries(ResizeModule opencv_core opencv_imgproc ModuleBase)
...
```

Implementation summary

Steps to create your own Module:

1. Copy .hpp and .cpp of an existing module e.g. the DummyModule
2. Replace "DummyModule" with your new name

Implementation summary

Steps to create your own Module:

1. Copy .hpp and .cpp of an existing module e.g. the DummyModule
2. Replace "DummyModule" with your new name
3. implement your logic in run()

Implementation summary

Steps to create your own Module:

1. Copy .hpp and .cpp of an existing module e.g. the DummyModule
2. Replace "DummyModule" with your new name
3. implement your logic in run()
4. define your module's metadata

Implementation summary

Steps to create your own Module:

1. Copy .hpp and .cpp of an existing module e.g. the DummyModule
2. Replace "DummyModule" with your new name
3. implement your logic in run()
4. define your module's metadata
5. edit CMakeList.txt

Implementation summary

Steps to create your own Module:

1. Copy .hpp and .cpp of an existing module e.g. the DummyModule
2. Replace "DummyModule" with your new name
3. implement your logic in run()
4. define your module's metadata
5. edit CMakeList.txt
6. run make

Implementation summary

Steps to create your own Module:

1. Copy .hpp and .cpp of an existing module e.g. the DummyModule
2. Replace "DummyModule" with your new name
3. implement your logic in run()
4. define your module's metadata
5. edit CMakeList.txt
6. run make

Live Demo

Live Demo!

Fork us on Github:

`https://github.com/TU-Berlin-CVRS/uipf`